# MODEL DOCUMENTATION

## Multi-Layer Perceptron (MLP)

### 1. Model Used

The model used in this project is a **Feedforward Neural Network (FNN)**, implemented using scikit-learn's MLPClassifier. It is a type of **Multi-Layer Perceptron (MLP)**, which is a supervised deep learning algorithm consisting of an input layer, one or more hidden layers, and an output layer

### 2. Why Feedforward Neural Network (FNN)?

**Captures Nonlinear Relationships:** Able to model complex, nonlinear interactions among features rather than relying on simple assumptions.

**Representation Learning:** Learns hierarchical feature representations from TF–IDF vectors, which enhances prediction quality.

**Flexible Architecture:** The number of layers and neurons can be customized to balance performance with computational cost.

**Generalization Power:** With techniques like dropout and early stopping, FNNs can adapt well to unseen test data.

**Scalable to Large Data:** More effective when working with large-scale datasets, making it suitable for balanced and imbalanced scenarios.

### 3. Working Principle

The Feedforward Neural Network (FNN) processes data in sequential layers:

1. **Input Layer**: Takes in the TF–IDF features of the text data.
2. **Hidden Layers**: Each neuron applies a weighted sum of inputs followed by a non-linear activation function (e.g., ReLU). This allows the network to capture complex patterns and relationships between words and ratings.
3. **Output Layer**: Uses a softmax activation function to generate probability scores for each rating class (1★ to 5★). The class with the highest probability is assigned as the prediction.

The model learns by adjusting weights through **backpropagation** and optimization (Adam solver in this case) to minimize the loss function.

### 5. Strengths of Feedforward Neural Network (FNN) in This Project

- Learns **nonlinear relationships** between features, going beyond simple assumptions.
- Works effectively with **TF–IDF features** by capturing richer interactions.
- Flexible architecture — hidden layers and neurons can be tuned for better performance.
- Provides **probabilistic outputs** via softmax, which can be interpreted as class confidence.
- Scales well to large datasets (e.g., 25,000 samples per class).

### 6. Limitations

- Requires **longer training time** compared to simpler models.
- Sensitive to hyperparameters (layers, neurons, learning rate, batch size).
- Less interpretable compared to simpler probabilistic models.
- Can **overfit** if not controlled with regularization techniques (e.g., early stopping, dropout).
- Performance depends heavily on consistent preprocessing (same TF–IDF vectorizer).

### 7. Hyperparameter Tuning

In this project, we performed hyperparameter tuning for the **FNN classifier** to balance training efficiency and accuracy.

- **Hyperparameters tuned**:
  - hidden_layer_sizes (number of layers and neurons per layer)
  - activation (ReLU chosen for efficiency and nonlinearity)
  - batch_size (controls training updates per step)
  - max_iter (maximum epochs to train)
  - early_stopping=True (to prevent overfitting)
- **Method used**: Manual experimentation with multiple architectures.
- **Best configuration found**:
  - Hidden layers: (256, 128)

    - Activation: ReLU
    - Batch size: 128
    - Early stopping enabled
- **Impact**: This configuration achieved around **61% accuracy** on the balanced dataset, significantly improving class-wise prediction consistency compared to simpler baselines.

**8. Conclusion**

In this project, a **Feedforward Neural Network (FNN)** was chosen because it provides a good balance between **accuracy, scalability, and computational efficiency** for text-based rating prediction tasks. While more advanced architectures like RNNs or Transformers could offer higher performance, FNN was selected due to its **simplicity, effectiveness with TF–IDF features, and suitability for large datasets**. All training, evaluation, and inference steps were carried out using FNN as the core deep learning model.

**1. Balance the Dataset**

```
# Balance the dataset to have equal number of reviews per rating

balanced_df = balance_dataset(review, target_col="Rating", n_samples=25000)
```

## 2. Train-Test Split & TF-IDF Vectorization

```
from sklearn.model_selection import train_test_split

from sklearn.feature_extraction.text import TfidfVectorizer


X = balanced_df['Review_text']

y = balanced_df['Rating']


# Stratified train-test split

X_train, X_test, y_train, y_test = train_test_split(

   X, y, test_size=0.2, stratify=y, random_state=42

)
```

# TF-IDF Vectorization

tfidf = TfidfVectorizer(max_features=25000, ngram_range=(1,2), stop_words='english')

X_train_tfidf = tfidf.fit_transform(X_train)

X_test_tfidf = tfidf.transform(X_test)

**3. Train Feedforward Neural Network (MLPClassifier)**

```
from sklearn.neural_network import MLPClassifier

# Initialize and train the model
mlp_model = MLPClassifier(hidden_layer_sizes=(256,128),
                          activation='relu',
                          solver='adam',
                          batch_size=128,
                          max_iter=20,
                          early_stopping=True,
                          verbose=True,
                          random_state=42)

mlp_model.fit(X_train_tfidf, y_train)
```

**4. Evaluate the Model & Hyperparameter Tuning**

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, classification_report

# Hyperparameter tuning for MLP
param_grid = {
```

```
    'hidden_layer_sizes': [(128,), (256,128), (512,256,128)],
    'batch_size': [64, 128],
    'max_iter': [20, 50]
}

grid = GridSearchCV(MLPClassifier(activation='relu', solver='adam', early_stopping=True, random_state=42),
                    param_grid,
                    cv=3,
                    scoring='accuracy',
                    verbose=2,
                    n_jobs=-1)

grid.fit(X_train_tfidf, y_train)

best_mlp_model = grid.best_estimator_
print("Best Parameters:", grid.best_params_)

# Evaluate tuned model
y_pred = best_mlp_model.predict(X_test_tfidf)
print("FNN Accuracy (tuned):", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```